Programação Orientada a Aspectos

António Menezes Leitão

23 de Abril de 2008

Pointcuts

Objectos

Representam todos os pontos de execução em que um objecto é um executor ou um receptor ou um argumento ou uma excepção.

Exemplo

- `this(Foo)`
  Todos os pontos de execução em que o `this` é uma instância da classe `Foo`.
- `target(Foo)`
  Todos os pontos de execução em que o `receptor` da invocação de um método é uma instância da classe `Foo`.
- `args(String,..,int)`
  Todos os pontos de execução em que o primeiro argumento é do tipo `String` e o último argumento é do tipo `int`.

Pointcuts

Captura de contexto

Idêntico ao anterior mas estes *pointcuts* podem capturar contexto.

Exemplo

- `this(x)`
  Todos os pontos de execução em que o `this` é uma instância do tipo do parâmetro `x`, associando esse parâmetro à instância.
- `target(x)`
  Todos os pontos de execução em que o `receptor` da invocação de um método é do tipo do parâmetro `x`, associando esse parâmetro ao receptor.
- `args(String,..,x)`
  Todos os pontos de execução em que o primeiro argumento é do tipo `String` e o último argumento é do tipo do parâmetro `x`, associando esse parâmetro ao último argumento.

Pointcuts

Invocação de método ou construtor

Representa um ponto de execução após a avaliação dos argumentos da invocação mas antes da execução do método invocado.

Exemplo

- `call(public void Foo.bar(Baz))`
  Todas as invocações do método público `bar` que tem um parâmetro do tipo `Baz` e que retorna `void` definido para a classe `Foo`.
- `call(Foo+.new(..))`
  Todas as invocações de construtores da classe `Foo` ou de qualquer sua subclasse.

Pointcuts

Execução de método ou construtor

Representa a execução de um método.

Exemplo

- `execution(public void Foo.bar(Baz))`
  Todas as execuções do método público `bar` que tem um parâmetro do tipo `Baz` e que retorna `void` definido para a classe `Foo`.
- `execution(Foo+.new(..))`
  Todas as execuções de construtores da classe `Foo` ou de qualquer sua subclasse.

Pointcuts

Acessos a *fields*

Representa leituras e escritas em *fields*.
O valor escrito pode ser capturado com o *pointcut* `args`.

Exemplo

- `set(int Foo.bar)`
  Todas as escritas no *field* `bar` de uma instância de `Foo`.
- `get(PrintStream System.out)`
  Todas as leituras do *field* `out` da classe `System`.

Pointcuts

Acessos a *fields*

```
aspect LimitedBarFieldChange {

    static final int MAX_BAR_CHANGE = 100;

    before(Foo foo, int newBar):
            set(int Foo.bar) &&
            target(foo) &&
            args(newBar) {
        if (Math.abs(newBar - foo.bar) > MAX_BAR_CHANGE) {
            throw new RuntimeException();
        }
    }
}
```

Pointcuts

Tratamento de excepcões

Representa a execução de um *exception handler* para um determinado tipo de excepção.
A excepção pode ser capturada com o *pointcut* `args`.

Exemplo

- `handler(IOException+)`
  Todas as execuções de blocos de um `catch` que lide com `IOException` ou qualquer das suas subclasses.
- `handler(Foo*)`
  Todas as execuções de blocos de um `catch` que lide com qualquer excepção cuja classe tem um nome começado por `Foo`.

Pointcuts

Tratamento de excepcões

```
aspect DumpFooException {

    before(FooException e):
            handler(FooException) && args(e) {
        e.printStackTrace();
    }
}
```

Pointcuts

Inicializações de classes e instâncias

Representa a execução das inicializações estáticas (*static blocks*) de classes ou das instâncias.

Exemplo

- `staticinitialization(Foo)`
  Execução de bloco estático da classe `Foo`
- `preinitialization(Foo.new(..)))`
  Todas as execuções da inicialização de uma instância quando o construtor é invocado, desde o início da execução do construtor da classe até ao início da execução do construtor da superclasse.
- `initialization(Foo.new(..))`
  Todas as execuções da inicialização de uma instância quando o construtor é invocado, desde o retorno da execução do construtor da superclasse até ao retorno da execução do construtor da classe.

Pointcuts

Estrutura léxica

Representa todos os pontos de execução dentro da estrutura léxica de uma classe (`within(TypePattern)`) ou método (`withincode(MethodOrConstructorSignature)`), incluindo classes internas (anónimas ou não).

Exemplo

- `within(Foo)`
  Todos os pontos de execução no âmbito léxico da classe `Foo`.
- `withincode(* Foo.bar(..))`
  Todos os pontos de execução no âmbito de qualquer método denominado `bar` da classe `Foo`.

Pointcuts

Fluxo de controle

Representa todos os pontos de execução no fluxo de controle de um dado ponto de execução.

Exemplo

- `cflow(call(* Foo.bar(..)))`
  Todos os pontos de execução no fluxo de controle da invocação de qualquer método `bar` da classe `Foo`, *incluindo* a invocação do próprio método.
- `cflowbelow(call(* Foo.bar(..)))`
  Todos os pontos de execução no fluxo de controle da invocação de qualquer método `bar` da classe `Foo`, *excluindo* a invocação do próprio método.

Pointcuts

Factorial

```
public class MyMath {

    int fact (int n) {
        if (n == 0) {
            return 1;
        } else {
            return n * fact (n - 1);
        }
    }

    public static void main(String[] args) {
        MyMath math = new MyMath();
        System.out.println(math.fact(10));
    }
}
```

Pointcuts

Aspecto

```
aspect LogFactorial {
    pointcut f(int i) :
        call(int fact(int)) && args(i);

    // most-recent
    before(int i, int j) :
        f(i) && cflowbelow(f(j)) {
        System.err.println(i + " previous " + j);
    }

    // original
    before(int i, int j) :
        f(i) && cflowbelow(cflow(f(j)) && !cflowbelow(f(int))) {
        System.err.println(i + " original " + j);
    }
}
```

Pointcuts

Resultado

```
9 previous 10
```

Pointcuts

Resultado

```
9 previous 10
9 original 10
```

Pointcuts

Resultado

```
9 previous 10
9 original 10
8 previous 9
8 original 10
```

Pointcuts

Resultado

```
9 previous 10
9 original 10
8 previous 9
8 original 10
7 previous 8
7 original 10
6 previous 7
6 original 10
5 previous 6
5 original 10
4 previous 5
4 original 10
3 previous 4
3 original 10
2 previous 3
2 original 10
1 previous 2
1 original 10
0 previous 1
0 original 10
3628800
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {







}
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {
    Object around(HttpRequest request) :




    }
}
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {
    Object around(HttpRequest request) :
            execution(* HttpServlet.do*(..)) && args(request,..) {





    }
}
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {
    Object around(HttpRequest request) :
            execution(* HttpServlet.do*(..)) && args(request,..) {
        long startTime = System.nanoTime();




    }
}
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {
    Object around(HttpRequest request) :
            execution(* HttpServlet.do*(..)) && args(request,..) {
        long startTime = System.nanoTime();
        Object retValue = proceed(request);




    }
}
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {
    Object around(HttpRequest request) :
            execution(* HttpServlet.do*(..)) && args(request,..) {
        long startTime = System.nanoTime();
        Object retValue = proceed(request);
        long endTime = System.nanoTime();



    }
}
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {
    Object around(HttpRequest request) :
            execution(* HttpServlet.do*(..)) && args(request,..) {
        long startTime = System.nanoTime();
        Object retValue = proceed(request);
        long endTime = System.nanoTime();
        monitorAgent.record(request.getRequestURI(),
                            endTime - startTime);

    }
}
```

Exemplo

Profiling

Registar o tempo que os *requests* levam a ser respondidos.

advice around

```
public aspect ServletPerformanceMonitor {
    Object around(HttpRequest request) :
            execution(* HttpServlet.do*(..)) && args(request,..) {
        long startTime = System.nanoTime();
        Object retValue = proceed(request);
        long endTime = System.nanoTime();
        monitorAgent.record(request.getRequestURI(),
                            endTime - startTime);
        return retValue;
    }
}
```

Exemplo (mau)

Factorial (errado)

```
public class MyMath {

    int fact (int n) {
        return -1;
    }

    public static void main(String[] args) {
        MyMath math = new MyMath();
        System.out.println(math.fact(10));
    }
}
```

Exemplo (mau)

Aspecto de correcção

```
aspect AroundFactorial {

    int around (MyMath mm, int n):
            call(int fact(int)) &&
            target(mm) &&
            args(n) {
        if (n == 0) {
            return 1;
        } else {
            return n * mm.fact(n - 1);
        }
     }
}
```

Aspectos Abstractos

AuthenticationAspect

```
abstract aspect AuthenticationAspect {

    abstract pointcut operationsNeeddingAuthentication();

    before() : operationsNeeddingAuthentication() {

        Authenticator.authenticate();
    }
}
```

DatabaseAuthenticationAspect

```
aspect DatabaseAuthenticationAspect extends AuthenticationAspect {

    pointcut operationsNeeddingAuthentication():
        call(* DatabaseServer.connect());
}
```

Aspectos Abstractos

Quais as invocações?

```
class Foo {
    void main(String[] args) {
        System.out.println(Foo.class);
        System.out.println(args[0] + args[1]);
    }
}
```

As aparências iludem

- `Foo.class` é implementado por intermédio de `Class.forName` protegido por um *exception handler* que apanha `ClassNotFoundException`.
- O operador `+` aplicado a `Strings` é implementado por invocações de `StringBuffer.append`.